See 1473

DDC (12)

DESIGN AND TESTING OF A GENERALIZED REDUCED GRADIENT CODE FOR NONLINEAR PROGRAMMING

BY

L.S. LASDON, A.D. WAREN, ARVIND JAIN and MARGERY RATNER

TECHNICAL REPORT SOL 76-3

FEBRUARY 1976

Systems Optimization Laboratory

Department of Operations Research

Stanford University

Stanford California 94305

DESIGN AND TESTING OF A GENERALIZED REDUCED GRADIENT CODE FOR NONLINEAR PROGRAMMING

by

L.S. Lasdon, A.D. Waren, Arvind Jain and Margery Ratner

TECHNICAL REPORT SOL 76-3

February 1976

SYSTEMS OPTIMIZATION LABORATORY

DEPARTMENT OF OPERATIONS RESEARCH

Stanford University
Stanford, California

Research and reproduction of this report were partially supported by the Office of Naval Research under Contracts N00014-75-C-0267, N00014-75-C-0865; U.S. Energy Research and Development Administration Contract E(04-3)-326 PA#18; the National Science Foundation Grant DCR75-04544 at Stanford University; and by the Office of Naval Research under Contract N00014-75-C-0240 and National Science Foundation Grant SOC74-23808 at Case Western Reserve University.

TABLE OF CONTENTS

1. Introduction

The purpose of this paper is to describe a Generalized Reduced Gradient (GRG) algorithm for nonlinear programming, its implementation as a FORTRAN program for solving small to medium size problems, and some computational results. Our focus is more on the software implementation of the algorithm than on its mathematical properties. This is in line with the premise that robust, efficient, easy to use NLP software must be written and made accessible if nonlinear programming is to progress, both in theory and in practice.

Recently, there has been increased emphasis on the software implementation of algorithms in many branches of mathematical programming, e.g., networks [1], mixed integer programming [2], unconstrained NLP [3, 4, 5] and some constrained NLP [6,7]. The earliest work on GRG is by Abadie [8,9], whose efforts form a basis for this work. Abadie and others [11, 12] have written GRG codes which have been disseminated to a limited extent. However, the detailed operation of these codes has not been described in the literature. We present some comparisons with Abadie's most recent code in Section 6 .

2. Brief Description of GRG Algorithms

GRG algorithms solve nonlinear programs of the form

$$\begin{array}{lll} \text{minimize} & g_{m+1}(X) & \\ \text{subject to} & g_i(X) = 0, & i = 1, \text{neq} \\ & 0 \le g_i(X) \le ub(n+i), & i = \text{neq} + 1, m \\ & lb(i) \le X_i \le ub(i) & i = 1, n \end{array} \qquad (1)$$

where X is a vector of n variables. The number of equality constraints, neq, may be zero. The functions g_i are assumed differentiable.

There are many possible GRG algorithms. Their underlying concepts are described in references [8] - [10]. This paper briefly describes the version currently implemented in our code.

The user submits the problem in the above form. It is converted to the following equality form by adding slack variables $X_{n+1}, \ldots, X_{n+m}$:

$$\begin{array}{lll} \text{minimize} & g_{m+1}(X) & \\ \text{subject to} & g_i(X) - X_{n+i} = 0, & i = 1, m \\ & lb(i) \le X_i \le ub(i), & i = 1, n+m \\ & lb(i) = ub(i) = 0, & i = n+1, n + \text{neq} \\ & lb(i) = 0 & i = n + \text{neq} + 1, n+m \end{array} \qquad (2)$$

These last two equations are the bounds for the slack variables. The variables $X_1, \ldots, X_n$ will be called "natural" variables.

Let $\bar{X}$ satisfy the constraints of (1), and assume that nb of the g_i constraints are binding (i.e., hold as equalities) at $\bar{X}$. A constraint g_i is taken as binding if

$$|g_i - ub(n+i)| < EPNEWT$$

or

$$|g_i - \ell b(n+i)| < EPNEWT$$

i.e., if it is within EPNEWT of one of its bounds. The tolerance EPNEWT is one of the most critical parameters in the code. It can be set by the user, and has a default value of 10^{-4}.

GRG uses the nb binding constraint equations to solve for nb of the natural variables, called the basic variables, in terms of the remaining n-nb natural variables and the nb slacks associated with the binding constraints.[1] These n variables are called nonbasic. Let y be the vector of nb basic variables and x the vector of n nonbasic variables, with their values corresponding to $\bar{X}$ denoted by $(\bar{y}, \bar{x})$. Then the binding constraints can be written.

$$g(y,x) = 0 \tag{3}$$

where g is the vector of nb binding constraint functions.[2] The basic variables must be selected so that the nb by nb basis matrix

---

[1] The degenerate case is considered in Section 5.6.

[2] The definitions of g are extended here to include the slacks.

$$B = (\partial g_i / \partial y_j)$$

is nonsingular at $\bar{X}$. Then the binding constraints (3) may be solved (conceptually at least) for y in terms of x yielding a function $y(x)$, valid for all (y,x) sufficiently near $(\bar{y},\bar{x})$. This reduces the objective to a function of x only

$$g_{m+1}(y(x), x) = F(x) \tag{4}$$

and reduces the original problem (at least in the neighborhood of $(\bar{y},\bar{x})$), to a simpler <u>reduced problem</u>

minimize $F(x)$

subject to $\ell \leq x \leq u$

where ℓ and u are the bound vectors for x. The function $F(x)$ is called the <u>reduced objective</u> and its gradient, $\nabla F(x)$, the <u>reduced gradient</u>.

This GRG code solves the original problem (1) by solving (perhaps only partially) a sequence of reduced problems. The reduced problems are solved by a gradient method. At a given iteration with nonbasic variables $\bar{x}$ and basic variables $\bar{y}$, B^{-1} is computed, and $\nabla F(\bar{x})$ is evaluated as follows:

$$\pi = (\partial g_{m+1}/\partial y)^T B^{-1}$$

$$\partial F/\partial x_k = \partial g_{m+1}/\partial x_k - \pi \, \partial g/\partial x_k$$

A search direction $\bar{d}$ is formed from $\nabla F(\bar{x})$ and a one dimensional search is initiated, whose goal is to solve the problem

$$\underset{\alpha>0}{\text{minimize}} \quad F(\bar{x} + \alpha\bar{d}) \quad .$$

This minimization is done only approximately, and is accomplished by choosing a sequence of positive values $\{\alpha_1, \alpha_2, \ldots\}$ for α. These are generated by subroutine SEARCH, described in Section 5.3. For each value α_i, $F(\bar{x} + \alpha_i\bar{d})$ must be evaluated. By (4), this is equal to $g_{m+1}(y(\bar{x} + \alpha_i\bar{d}), \bar{x} + \alpha_i\bar{d})$, so the basic variables $y(\bar{x} + \alpha_i\bar{d})$ must be determined. These satisfy the system of equations

$$g(y, \bar{x} + \alpha_i\bar{d}) = 0 \qquad (5)$$

where $\bar{x}$, $\bar{d}$, α_i are known and y is to be found. This system is solved by a variant of Newton's method.

The one dimensional search can terminate in three different ways. First, Newton's method may not converge. If this occurs on the first step, i.e., $i = 1$ in (5), then α_1 is reduced and we try again. Otherwise, the search is terminated.[1] Second, if the Newton method converges, some g_i constraints (which were previously not binding) or basic variable bounds may be violated. Then the code determines a new α value such that at least one such new constraint or variable is at its bound and all others are

[1] For a discussion of alternative strategies, see Section 5.3.

satisfied. If certain conditions are met (see description of subroutine SEARCH, Section 5.3), the new constraint is added to the set of binding constraints, the one dimensional search is terminated, and solution of a new reduced problem begins. Finally, the search may continue until an objective value is found which is larger than the previous value. Then a quadratic is fit to the three α_i values bracketing the minimum, F is evaluated at the minimum of this quadratic, and the search terminates with the lowest F value found. The reduced problem remains the same.

An important feature of this algorithm is its attempt to return to the constraint surface at each step in the one dimensional search. This differs from earlier strategies suggested by Abadie [9] and by Luenberger [13], which involve linear searches on the tangent plane to the constraint surface prior to returning to that surface. We have not experimented with such strategies, choosing to return to the surface each time because it was simpler and we felt it would lead to a more reliable algorithm. Computational experience presented later shows that if properly implemented, this strategy can be developed into an algorithm that is both reliable and efficient.

3. Program Structure

This code is composed of a main program and a number of subroutines, written in FORTRAN IV. The main program and subroutines are arranged in a hierarchical structure, as shown in Figure 1. (Subroutines of minor importance are not shown.)

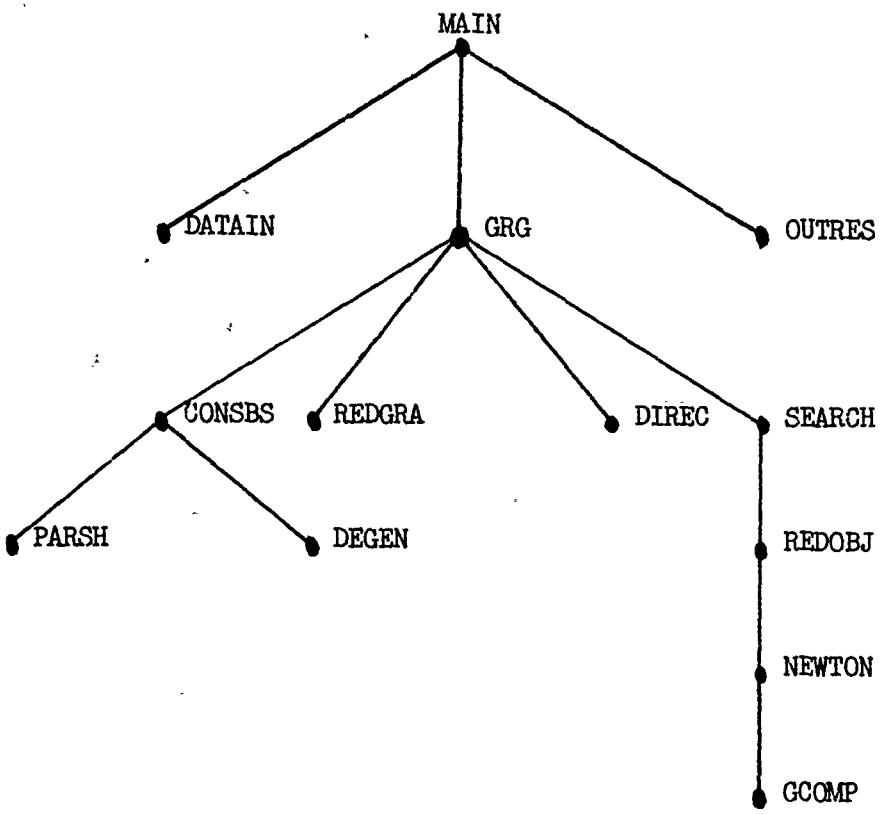


Figure 1. GRG Main Program and Subroutines

The primary function of MAIN is to call DATAIN to read the input data, GRG to solve the problem, and OUTRES to print the results. It then tests a user-specified flag to decide whether to stop or to return to read additional data. It is very simple, and is easily modified to work within a larger system. All subroutines below and including GRG comprise the GRG algorithm for iteratively solving the problem (1)-(2). Their functions are briefly described in Table 1.

1.	GRG	Controls main iterative loop. Computes initial basis inverse and calls DIREC to compute search direction. Calls one dimensional search subroutine SEARCH. Tests for optimality.
2.	CONSBS	Computes Basis Inverse, BINV.
3.	DEGEN	Computes search direction when basis is degenerate.
4.	PARSH	Given current X and G vectors, computes array GRAD, whose (i,j) element is the partial derivative of g_i with respect to X_j. May be user supplied. If not, there is a system subroutine PARSH which computes GRAD by forward difference approximation.
5.	REDGRA	Given BINV and GRAD, computes Lagrange multiplier vector π, and reduced gradient of either phase I or phase II objectives, GRADF.
6.	DIREC	Computes search direction.
7.	SEARCH	Performs one dimensional search.
8.	REDOBJ	Computes values of basic variables for given values of nonbasics by calling NEWTON. Takes action if NEWTON does not converge. Checks for constraint violations. If any are violated, finds feasible point where some initially violated constraint is binding and others satisfied.
9.	NEWTON	Uses Newton's Method to compute values of basic variables for given values of nonbasics. If convergence not achieved, sets flag and returns.
10.	GCOMP	User supplied subroutine. Given current X vector, computes vector of m+1 function values G, where G(1), ..., G(m) are constraint function values and G(m + 1) is the objective.

Table 1. Functions of Major GRG Subroutines

4. User Inputs

The only subroutine which the user must provide is GCOMP, which computes the function $g_1, \ldots, g_{m+1}$ for a given vector X. First derivatives of the functions g_i are required, but these may be computed by a system subroutine PARSF using finite difference approximations (first order forward differencing). Alternatively, the user may supply a subroutine PARSF which computes first derivatives by analytic formulas or other means. The finite difference approximations have been completely satisfactory in all problems solved thus far, and eliminate the (often considerable) burden of coding PARSF.

The other input data are the problem dimensions m, n, neq, the bounds ub_i and lb_i, initial values for X, and (optionally) a list of variables to be in the initial basis if that is possible. If the initial values of X do not satisfy the bounds (2), an error message is printed and the program stops. There is no conceptual difficulty in augmenting the phase I procedure to deal with bound violations, and future versions will include this feature. All data are checked for obvious errors (e.g., $ub_i < lb_i$) and are printed for inspection. In addition to the above mentioned data, the user may specify a number of control and tolerance parameters (to be discussed later), or may leave them at their default values.

5. Detailed Algorithmic Structure

The flow charts in this section are in aggregated form. Their purpose is to describe overall program logic. However, they correspond closely to the actual FORTRAN code.

5.1 Subroutine GRG

This is the controlling subroutine for the iterative process. It starts by calling GCOMP to evaluate the initial constraint and objective values. If any constraints are violated, a phase I procedure is entered in which the objective is the sum of absolute values of constraint violations. The array ICAND in block 1 is used by CONSBS in choosing basic columns. CONSBS will attempt to enter columns in ICAND into the basis first (in the order listed) before trying other columns. If the user has not specified an initial ICAND, block 1 sets it to the index set {1, .., n}.

The current X is considered optimal if either of two tests is met in block 2. The first test checks if the following conditions are met:

for i = 1,n but x(i) not a slack variable for an equality constraint

$$x(i) = lb(i) \implies GRADF(i) \geq -EPSTOP$$

$$x(i) = ub(i) \implies GRADF(i) \leq EPSTOP$$

$$lb(i) < x(i) < ub(i) \implies |GRADF(i)| < EPSTOP$$

The quantities x(i) are the current nonbasic variables and GRADF(i)

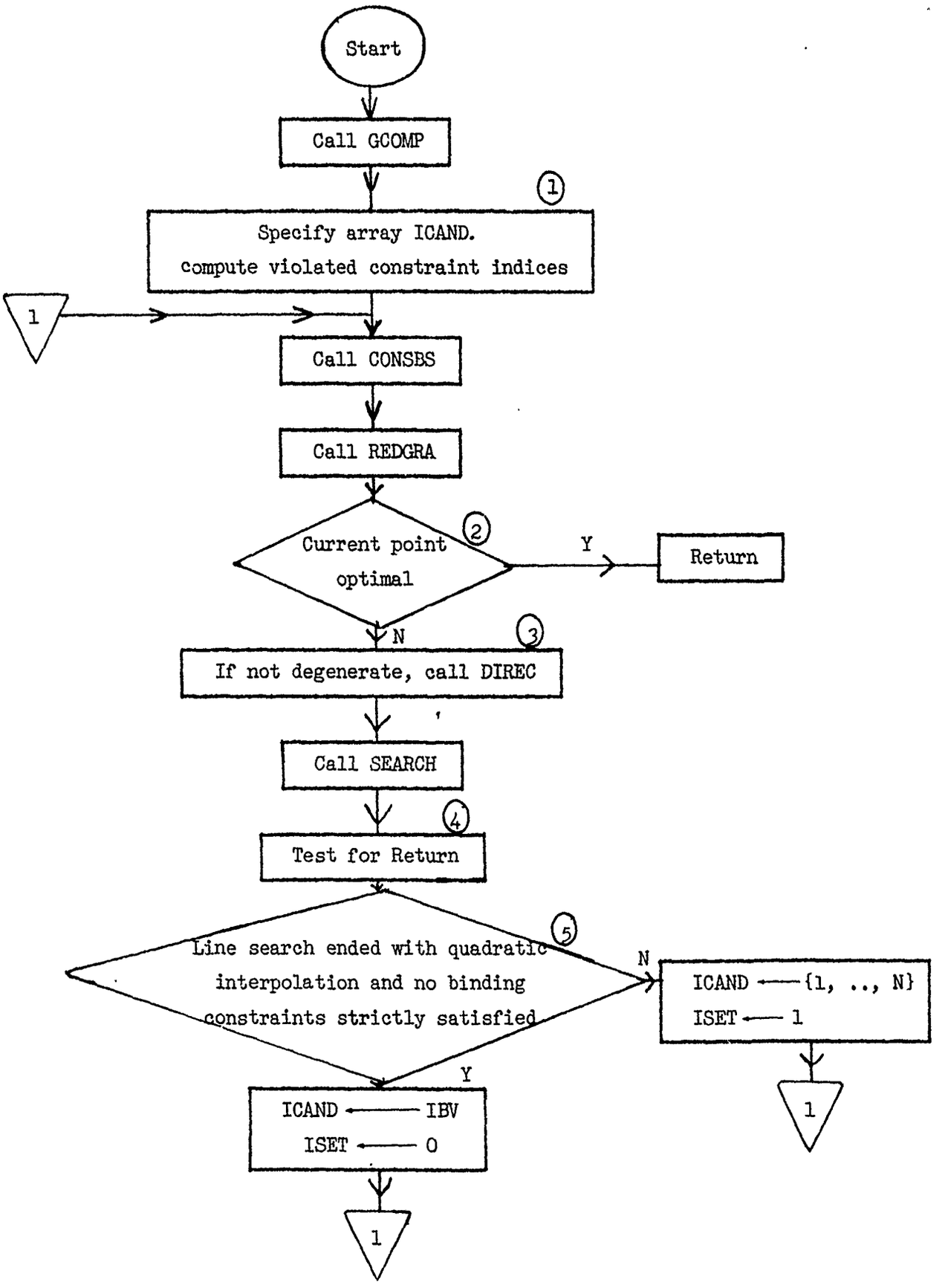
is the i^{th} component of the reduced gradient. This tests whether the Kuhn-Tucker optimality conditions are satisfied to within EPSTOP, a small positive num' which can be controlled by the user, with default value 10^{-4}. The slack variables for equality constraints (i.e., the variable X(n + 1) to X(n + neq)) are excluded from the test because they must be zero in any feasible solution. The second optimality test checks if the condition

ABS(FM - OBJTST) < EPSTOP * ABS(OBJTST)

is satisfied for NSTOP consecutive iterations. In the above, FM is the current objective value and OBJTST is the objective value at the start of the previous one dimensional search. NSTOP has a default value of 3.

If CONSBS constructs a degenerate basis then it also computes a useable feasible search direction. Hence, DIREC need not be called in block 3. Otherwise, DIREC generates a search direction and, in either case, SEARCH is called to find the best point in the given direction.

Figure 2. Subroutine GRG



The test in block 4 returns if SEARCH has halved the initial step size 10 times without finding a lower objective value and if $d = -\nabla F$. A small fraction of problems have terminated in this way, always with objective values close to optimality. In block 5, the Y branch indicates the case where the basis is unchanged and the line search terminated as in an unconstrained problem. Then ICAND is set to the previous basic variable list, IBV, so that CONSBS will again choose these as basic variables, if possible. This is desirable, because of the use of the BFS variable metric algorithm [14] in DIREC to choose the search direction, d. This algorithm uses an approximation to $(\nabla^2 F(x))^{-1}$ to generate d, and the approximation is not valid unless the reduced problem remains the same. If the reduced problem changes (N branch, block 5), ICAND is set to {1, ..., N} to give CONSBS freedom to construct as well conditioned a basis as possible. Setting ISET to 1 forces DIREC to reset the BFS algorithm so that $d = -\nabla F$. Testing for strictly satisfied constraints in block 5 permits reducing the size of the basis by dropping these from the binding set.

In most problems solved thus far, initial iterations end with a new constraint being added to or deleted from the basis or with a change in basic variables. The code is finding the set of constraints which are binding at optimality, and a compatible set of basic variables. Once this is done, the BFS algorithm takes over, and the code operates in an "unconstrained" mode until the optimum is reached. Most of the improvement in the objective and most of the computations occur in the first phase of this process, so it must be done efficiently. The critical operations

here involve violating one or more non-binding constraints during the line search, and finding a new feasible point where one (or more) of the violated constraints is binding. This latter operation occurs in subroutine REDOBJ.

5.2 Subroutine DIREC

This subroutine computes a search direction, d. Currently d is computed using the Broyden-Fletcher-Shanno (BFS) variable metric algorithm [14]- [16], modified to accomodate upper and lower bounds on the variables as suggested by Goldfarb [17]. An $n \times n$ matrix H, which is updated after each step, is used to determine d as

$$d = -H * GRADF \qquad (6)$$

where GRADF is the reduced gradient. If no one-dimensional searches have been performed or subroutine GRG has set ISET to 1, then d is taken as the negative reduced gradient direction and H is reset to an identity matrix with zero diagonal elements for nonbasic variables at bounds. H is updated only if a test is passed (see [16]) indicating that the new H will be positive definite. Numerical error may invalidate this test, so the directional derivative $r = \nabla F^t d$ is tested after applying (6); if $r \geq 0$, H is reset as described above.

The BFS method was chosen because computations in [14] - [16] and elsewhere show it to be one of the most efficient and reliable of the variable metric algorithms. In the absence of rounding error, it minimizes a quadratic function of n variables in at most n one dimensional searches. Hence, this GRG algorithm is finite for quadratic programs.

The last function of DIREC is to test whether nonbasic variables currently at bounds should be allowed to move away from these bounds. This is done as follows: the signs of the reduced gradient components of nonbasic variables at bounds are checked. Among all variables at lower (upper) bounds whose reduced gradient components are negative (positive), the gradient component of largest absolute value is chosen. If this value is denoted by MAXMULT and the corresponding index is IDROP, then variable IDROP is released from its bound if

$$\sum_{i=1}^{N} (d(i))^2 < (\text{MAXMULT})^2/4$$

This test, prescribed in [17], is designed to prevent "zigzagging", i.e., variables continually coming on and off their bounds.

Since H is a symmetric matrix, only the diagonal and super-diagonal elements are needed, and these are stored in a linear array H in row order. Storage requirements can be further reduced by noting that H has zero rows and columns corresponding to nonbasic variables at their bounds. This feature, not yet exploited, would significantly reduce

computation time and storage requirements when most nonbasic variables are at bounds. An extension of this idea to general linear constraints has been developed by Prof. A. Buckley [18], who pointed out its implications to us. Of course, H can be eliminated entirely by use of the Conjugate Gradient Algorithm [19], which is also finite for quadratic functions. Its use will be examined in future GRG codes designed for large scale problems.

Localizing all computation of the search direction in a single subroutine provides a great deal of flexibility. Other direction finding procedures can be implemented simply by coding a new DIREC, leaving the rest of the system unchanged.

5.3 Subroutine SEARCH

Subroutine DIREC provides search directions for subroutine SEARCH, in which the variables of the problem are assigned new values. This subroutine finds a first local minimum for the problem

$$\underset{\alpha}{\text{minimize}}\ F(\bar{x} + \alpha d)$$

The direction d is always a direction of descent, i.e.,

$$d^T \nabla F(\bar{x}) < 0$$

This subroutine searches for three α values, A, B and C, which satisfy

$$0 \leq A < B < C$$

$$F(\bar{x} + Ad) > F(\bar{x} + Bd) \leq F(\bar{x} + Cd) .$$

Then the interval $[A,C]$ contains a local minimum of $F(\bar{x} + \alpha d)$. In block 11 of Figure 3, a quadratic in α is passed through A, B, and C, with its minimum at D. The best point, B or D, is taken as an estimate of the optimal α and a return is made.

In finding (A,B,C) the choice of initial step size, B, (block 1), is important. With the BFS or other variable metric methods, B is set equal to the optimal α value from the previous search except when this causes too large a change in the variables. The theoretical

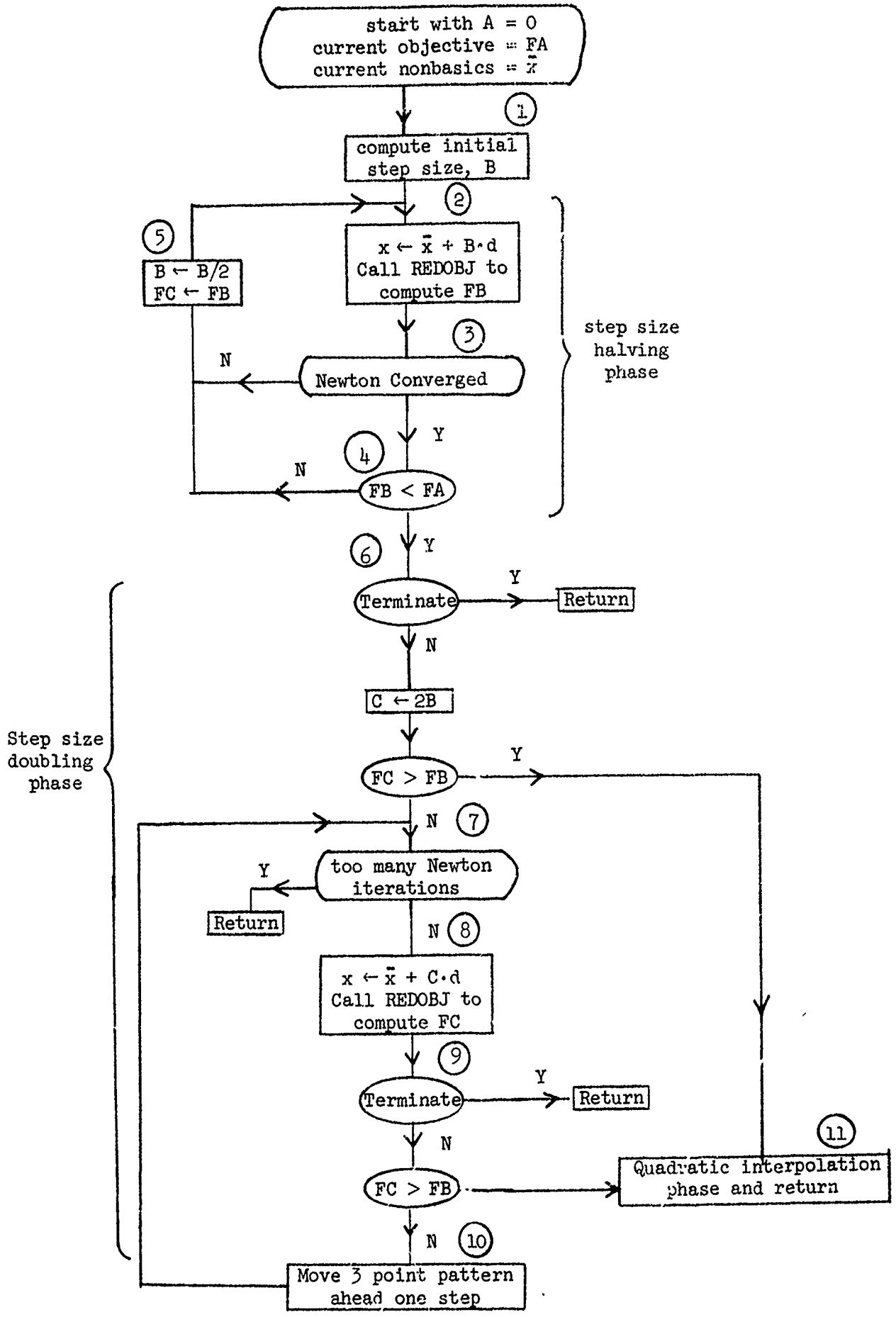


Figure 3. Subroutine Search

basis for this is that, as a variable metric method converges, the optimal α values should converge to 1, the optimal step size for Newton's Method. Hence, the previous optimal step is a good approximation to the current one. This must be modified when the method is restarted, for example, when a new constraint is encountered or the basis is changed, since then an optimal step much less than unity is generally taken. Hence, we require that the change in any nonbasic variable larger than 1.0 in absolute value not exceed .05 times its value, while the change in any variable smaller than 1.0 in absolute value cannot exceed 0.05. If the largest α value meeting these conditions is α^1, and α_{i-1} is the step size found by SEARCH at iteration i-1, then

$$B = \min(\alpha_{i-1}, \alpha^1)$$

if the previous search terminated with an interpolation, and $B = \alpha^1$ otherwise.

The subroutine operates in two phases, halving the initial step size (if necessary) until an improved point is found (loop 2 - 3 - 4 - 5), and doubling the step size until the minimum is bracketed (loop 7 - 8 - 9 - 10). In each phase the nonbasic variables are changed, and subroutine REDOBJ is called to compute the reduced objective $F(\bar{x} + \alpha d)$ (blocks 2 and 8). If, in the doubling phase, the minimum is not bracketed, the A and B points are replaced by the B and C points, C is replaced by 2B (block 10), and the new F value is computed.

The only blocks which would not appear in a line search for unconstrained problems are 3, 6, 7, and 9. These deal with two occurrences: (a) In attempting to evaluate the basic variables in subroutine REDOBJ by Newton's method, Newton may not converge, and (b) REDOBJ may produce a point at which a previously loose constraint is binding. In block 3, the step size is halved if Newton does not converge. Theoretically, convergence must ultimately occur; in practice, if the loop 2 - 3 - 4 - 5 is traversed 10 times, a return to GRG is made. Blocks 6 and 9 terminate the search if case (b) occurs, while 9 also terminates under case (a). Block 7 terminates the search if more than 6 Newton iterations were required the last time the basic variables were evaluated in subroutine NEWTON. Experience has shown that the next NEWTON call usually will not converge.

Before beginning the line search, SEARCH computes the largest α, $\bar{\alpha}$, such that $\bar{x} + \alpha d$ satisfies the nonbasic variable bounds. The logic required to insure that α does not violate this bound, and to detect the case where $\bar{\alpha}$ is optimal, has been omitted from the flow chart for simplicity.

The strategy of terminating the search if an improved point has already been found and Newton did not converge or took too many iterations was not adopted initially. Earlier versions of the code attempted to push on with the line search by recomputing B^{-1} at the last feasible point found and cutting the step taken to 1/3 its previous value. The current strategy has been found to be much superior. Now B^{-1}

is computed only once, at the beginning of each one dimensional search, where it is needed anyways to compute ∇F. In a set of six test problems, the current strategy required about half the function and gradient evaluations of the older one (see [20] for details), and also significantly reduced the number of times B^{-1} is evaluated. This strategy, plus the rather conservative choice of initial step size, plays a large part in making an algorithm which returns to the constraint surface each time efficient.

Recently, much interest has been expressed in "step size" algorithms to replace the line search. These attempt to obtain only a "sufficient" decrease in the objective, rather than to find its minimum. We have opted for a "sloppy" attempt to find the minimum because of its "inside-out" nature, i.e., a small initial step is taken, then increased if necessary. This approach is more likely to remain within the radius of convergence of Newton's method, viz., that range of α values for which Newton will converge, with B^{-1} evaluated only at $\alpha = 0$. This radius imposes an upper bound on α of unknown value. Most step-size strategies choose large α values first (e.g., $\alpha = 1$ in [22]), then reduce them if necessary, and hence are more likely to exceed this bound. The fact that the number of Newton iterations required to evaluate $F(x + \alpha d)$ increases as α increases also works against "outside-in" strategies. These same comments also apply to algorithms which do a one dimensional search on the tangent plane to the constraints prior to returning to the constraint surface, as in [11]-[13].

5.4 Subroutine REDOBJ

This subroutine evaluates the reduced objective function $F(\bar{x} + \alpha d)$ for given $\bar{x}$, α, and d. It does so by attempting to solve the system of nb (possibly nonlinear) equations

$$g_i(y, \bar{x} + \alpha d) = 0 \qquad i \in IBC \qquad (7)$$

for the nb basic variables y, where IBC i the index set of binding constraints. As in [8] and [9], this is accomplished in subroutine NEWTON, using the pseudo-Newton algorithm

$$y_{t+1} = y_t - B^{-1}(\bar{X})\, g_B(y_t, \bar{x} + \alpha d), \qquad t = 0, 1, 2, \ldots \qquad (8)$$

where g_B is the vector of binding constraints. The algorithm is called pseudo-Newton because B^{-1} is evaluated once at the initial point of the search, $\bar{X}$, instead of being reevaluated at each step of the algorithm, as in the standard Newton method.

An initial estimate of the basic variables, y_0, is computed either by linear or quadratic extrapolation in block 1 of Figure 4. As in [8], the linear extrapolation uses the tangent vector

$$v = -B^{-1}\left(\frac{\partial g}{\partial x}\right) d \qquad (9)$$

In our code, v is computed at $\bar{X}$. It is used to find initial values, y_0, by the formula

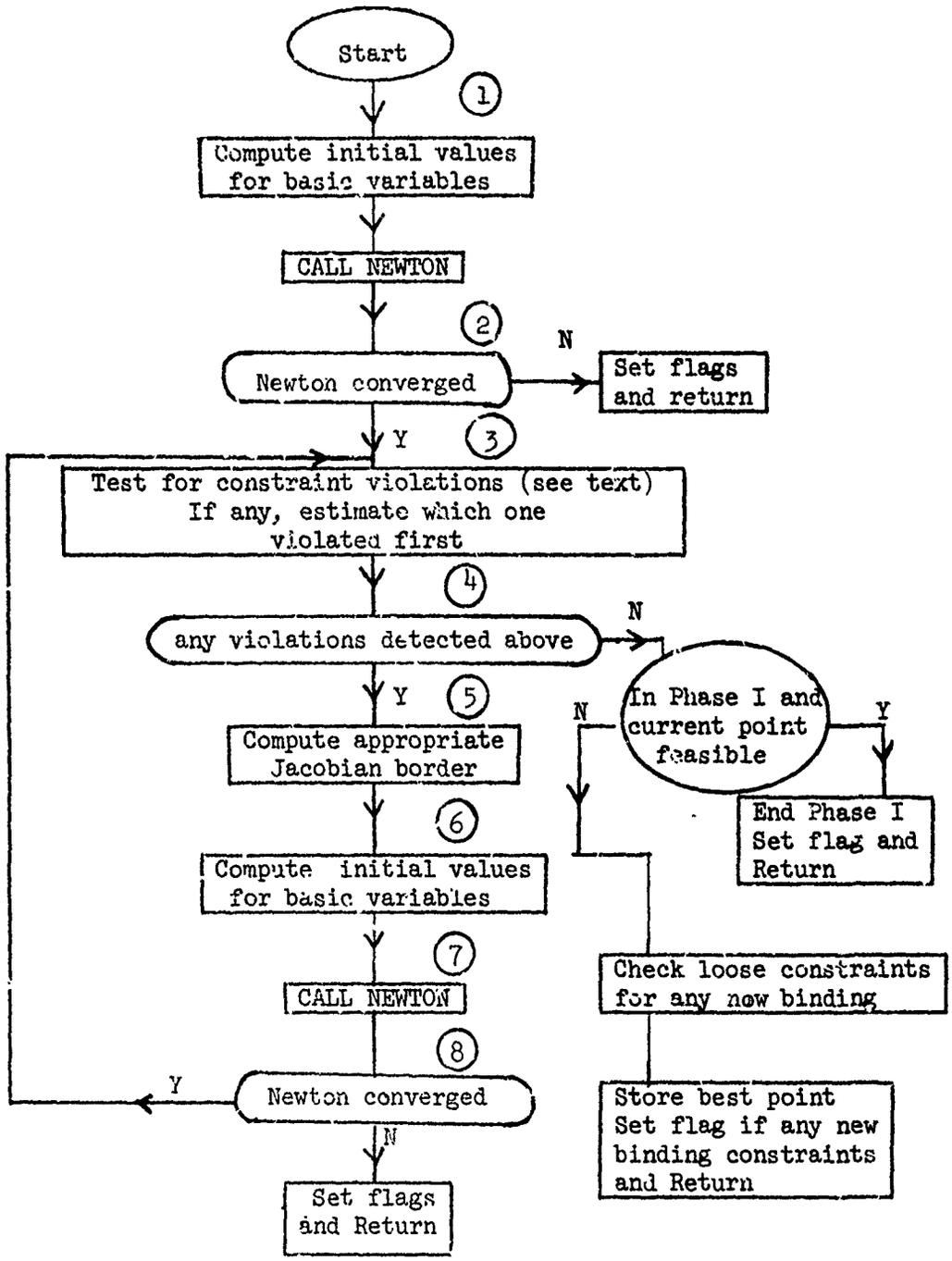


Figure 4. Subroutine REDOBJ

$$y_0 = \bar{y} + \alpha_1 v$$

Using these initial values, Newton finds the feasible point X_1. Then, at X_1, v is not recomputed. The old v is used, but emanating now from X_1, to yield the next set of initial values as

$$y_0 = y_1 + (\alpha_2 - \alpha_1)v$$

Using these, Newton finds a new point X_2. This procedure is repeated until the one dimensional search is over.

Quadratic extrapolation may also be used to obtain initial estimates of the basic variables, at the users option. Initially, linear extrapolation is used. After the first feasible point, X_1, is found, quadratic functions are fit to the value and slope at $\alpha = 0$, and the value at α_1 of each basic variable. These are used to predict their values at $\alpha = \alpha_2$. Subsequent quadratics are fit to the values of the basic variables at α_i, α_{i-1}, α_{i-2} $(i \geq 2)$. Computational experience with these options appears in Section 6.4.

In blocks 2 and 8, Newton is considered to have converged if the condition

$$\text{NORMG} = \max_{i \in \text{IBC}} |g_i(X_t)| < \text{EPNEWT}$$

is met within ITLIM iterations. Currently EPNEWT = 10^{-4} and ITLIM = 10. If NORMG has not decreased from its previous value (or the above condition is not met in 10 iterations) Newton has not converged.

Once Newton has converged, possible constraint violations must be checked (block 3). There are several reasons why the current step α may be too large;

(1) A strictly satisfied constraint may have violated an upper or lower bound.

(2) A constraint in IABOVE, the set of constraints initially violating their upper bounds, may violate a lower bound.

(3) A constraint in IBELOW may violate an upper bound.

(4) A basic variable may violate a lower or upper bound.

If any of these cases hold, α is reduced to a value α^* where no constraints are violated and at least one new constraint is equal to a bound. To determine this constraint, an estimate is made of α^* in block 3. Linear interpolation between the current and previous values of the violated constraint is used.

The next step determines whether case 4 or one of cases 1 - 3 is to be dealt with, according to which has the smallest linear estimate of α^*. Assuming cases 1 - 3 as an example, we then wish to solve the system

$$g_i(y(\alpha), \bar{x} + \alpha d) = 0, \qquad i \in IBC$$

$$g_L(y(\alpha), \bar{x} + \alpha d) = 0 \tag{10}$$

where α is a new variable and $g_L = 0$ a new equation. The Jacobian for this system is

$$J = \left[\begin{array}{c|c} B & c \\ \hline d & w \end{array}\right]$$

where

$$d = \partial g_L / \partial y$$

$$w = (\partial g_L / \partial x)^T d$$

and c is an nb component column vector whose elements are $(\partial g_i / \partial x)^T d$ for $i \in$ IBC. The border vectors c, d, and the scalar w are computed in block 5. Until recently, this block also computed J^{-1} by the bordered inverse formula (using the known B^{-1}), and this J^{-1} was used by subroutine NEWTON in block 7 to solve the augmented system (10). However, this has the disadvantages that the old B^{-1} is erased, and it would not be suitable for a large scale GRG implementation since it involves changing the dimension of the inverse. However, inspection of (8) shows that J^{-1} is not required, only the product of J^{-1} with the vector (g_B, g_L). This product can be expressed in terms of B^{-1} and the border elements c, d, w using the bordered inverse formula. This is done in subroutine NEWTON, which tests to see if the system (7) or the augmented system (10) is to be solved, and computes the appropriate matrix-vector product.

The basis change procedure here, based on solving (10) and then calling CONSBS, differs from that of Abadie in [8] and [9].

In Abadie's procedure, the basic variables y_t in (8) are checked for each t. If any violate their bounds, one is selected to leave the basis, an entering variable is chosen, and a pivot operation updates B^{-1}. The Newton iteration (8) then continues. This could lead to "false" basis changes, since violation of a bound during the iterative process does not imply violation when the process converges. Newton's method need not converge componentwise monotonically. Failure to converge could also lead to a false basis change. In addition, if the (Abadie) basis change is accomplished, the new point may have an objective value larger than $F(\bar{x})$, which makes proof of convergence unlikely. Our procedure avoids both these objections, perhaps at the expense of slightly more computation. We feel that the (presumed) increase in reliability makes the tradeoff worthwhile.

5.5 Subroutine CONSBS

This subroutine selects a set of basic variables and computes the basis inverse, B^{-1}. Its input is a list of indices of variables, the candidate list ICAND. The outputs of CONSBS include (a) a new list of binding and strictly satisfied constraint indices (b) a new list of basic and nonbasic variable indices and (c) the new basis inverse, B^{-1}.

The call to PARSH in block 1 of Figure 5 computes the gradients of the objective and all constraints at the current point. The gradients of the binding constraints are stored in an array, which is where the pivoting to compute B^{-1} is done.

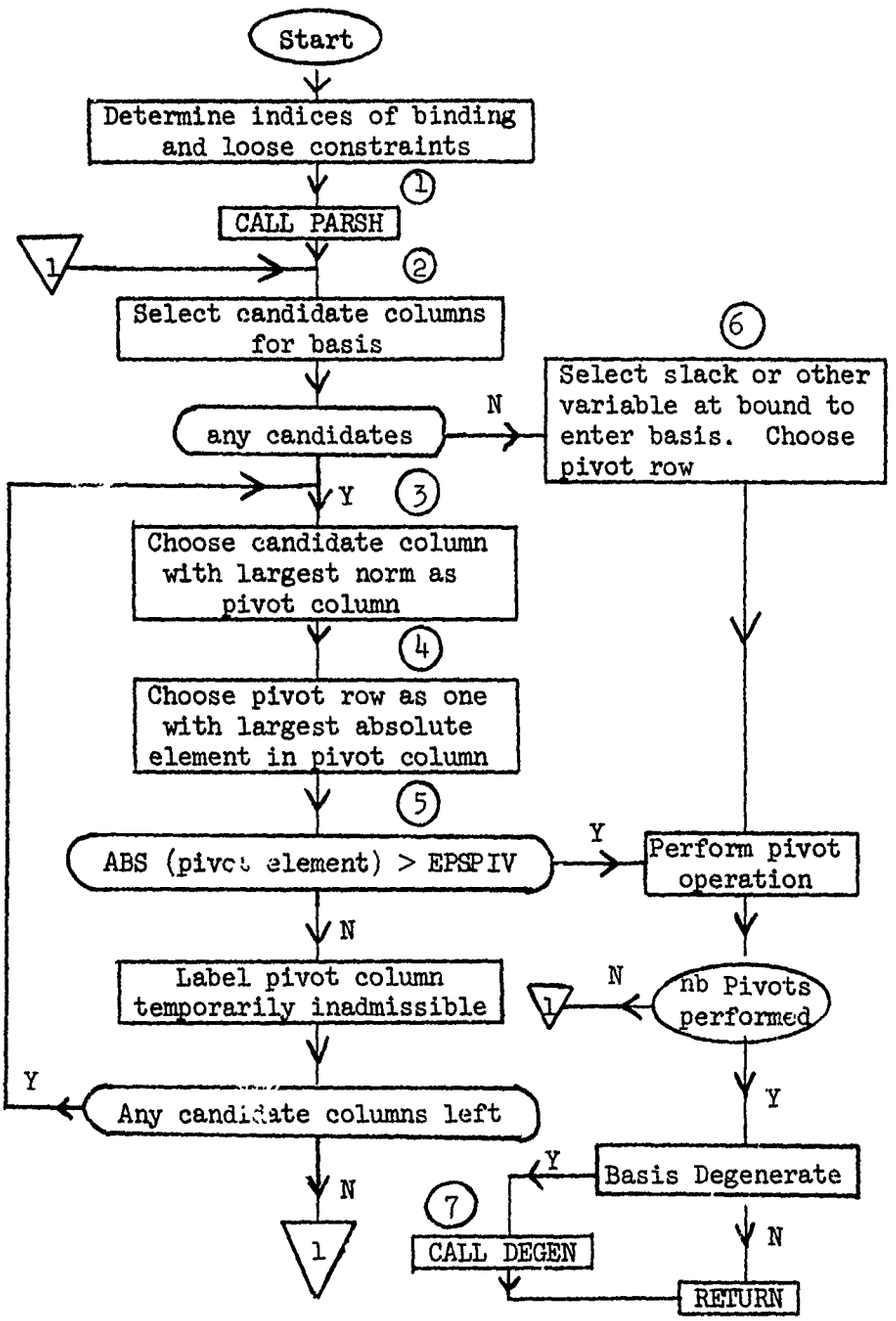


Figure 5. Subroutine CONSBS

The subroutine operates in 2 modes.* In mode 1 CONSBS will choose pivot columns from whatever candidate list was input to it. If a basis inverse could not be constructed from columns in this candidate list, or if the original candidate list included all variables, the mode indicator is set to 2, and CONSBS will choose pivot columns from the list of all admissible columns. A column is admissible if it is not scheduled to leave the basis and if it has not yet been pivoted in. For simplicity, the mode logic is not shown in Figure 5.

In block 2, candidate columns are chosen as those which are admissible and whose variables are farthest from their nearest bound (to try to reduce the number of basis changes required). A maximum of five are chosen, and variables very close to their nearest bound are ignored. If no candidates can be found and we are in mode 1, the mode is set to 2, the candidate list becomes the set of all admissible columns, and we return to point (1). If we are in mode 2, a variable at a bound is chosen to enter the basis (block 7). The basis is labeled degenerate. A pivot row is chosen in the same way as when the basis is nondegenerate, so we proceed to discuss that case.

The logic in blocks 3 and 4 represents complete pivoting within the set of candidate columns, i.e., the entire submatrix is examined to find the largest pivot element. In block 3, the L_1 norm is used, taken over all rows not yet pivoted in. The default value of EPSPIV in block 5 is 10^{-3}.

If the basis formed is degenerate, subroutine DEGEN is called in block 7 to compute a search direction which will decrease the objective without immediately violating any basic variable bounds. A flag is set

*We thank Professor Jean Abadie for suggesting that we design CONSBS in this way.

to bypass the call to DIREC in subroutine GRG. DEGEN is discussed in the next section.

Earlier versions of CONSBS employed partial pivoting, i.e., the pivot row was chosen in sequential order 1, 2, ... , NB, while the pivot column was chosen as the column selected by block 3 with largest absolute element in that row. This was replaced by complete pivoting when it failed on some test problems, encountering rows where all elements were too small, and where a nonsequential order of choosing rows led to acceptable pivots.

A recent enhancement to CONSBS consists of additional logic just prior to entering block 6. At this point, all admissible pivot elements corresponding to variables not at bounds have absolute values smaller than EPSPIV (default value 10^{-3}). The new logic tests the largest admissible pivot element; if its absolute value is larger than 10^{-2} * EPSPIV, we choose it as the pivot element rather than accepting a variable at a bound. The motivation for this is that pivoting on an element larger than 10^{-5} is not likely to cause severe problems, especially if it is the largest element available, and if the alternative is introducing a variable at a bound into the basis. This new logic has permitted solution of a few test problems which could not be solved previously. The previous failures were caused by very small decreases in objective value, along (unit vector) search directions generated by subroutine DEGEN. With the new logic, DEGEN is not called, and the search directions generated by DIREC lead to the optimum.

5.6 Subroutine DEGEN

Subroutine DEGEN is called by CONSBS when the basis constructed is degenerate, i.e., has one or more basic variables at bounds. In this case, the search direction, d, produced by DIREC may cause some of these variables to violate a bound immediately,i.e., d may not be a feasible direction. DEGEN computes a useable feasible direction or proves that the current point satisfies the Kuhn-Tucker conditions.

Let L and U be the index sets of basic variables at lower and upper bounds respectively. A direction d is feasible if the following conditions are satisfied:

$$Bv + Nd = 0 \tag{11}$$

$$v_i > 0, \quad i \in L \tag{12}$$

$$v_i < 0, \quad i \in U \tag{13}$$

Here B is the basis, and N contains the nonbasic Jacobian columns. The vector v is the tangent vector of equation (9), and is the vector of directional derivatives of y in the direction d. Conditions (12)-(13) require that basic variables at lower bound increase, and conversely for variables at upper bound. Strict inequality is required in (12) and (13) to ensure that v_i remains non-negative for a small movement in the direction d. Our code relaxes conditions (12)-(13) to $v_i \geq -\epsilon$, $i \in L$ and $v_i \leq \epsilon$, $i \in U$, with ϵ in the range 10^{-5} to 10^{-7}. This accepts a small amount of infeasbility, allowing for numerical error.

DEGEN begins by checking if the direction d produced by DIREC is feasible. If so it is used. Otherwise the column of N whose reduced gradient yields the largest rate of improvement in F is found--if none yield improvement, the Kuhn-Tucker conditions are satisfied and the current point is accepted as optimal. The direction d is set to the corresponding unit vector and if v (equal to B^{-1} times the column chosen) passes the feasibility test, d is accepted. Otherwise the largest absolute element of v in rows with indices in L or U is found. If it is large enough, it is used as a pivot element, a pivot operation is performed in (11), and the process begins again with the new basis B. If no basis is repeated this process must terminate finitely, either with a useable feasible direction or with the optimality test met. In the 3 or 4 degenerate problems solved thus far, calls to DEGEN have required at most 2 or 3 pivots.

The test referred to above for a pivot element being large enough initially uses a tolerance of 10^{-3}. If a potential pivot element is less than this, that column is labelled temporarily inadmissible. If all improving columns become inadmissible, the pivot tolerance is reduced to the same ϵ value used to test for feasibility, and the inadmissible column with largest pivot element is entered into the basis.

6. Computational Experiments

6.1. Comparison with Interior Penalty Methods

Seven test problems were solved by the GRG code and by the interior penalty code described in [21]. Problem characteristics are shown in table 2

Problem	Source or Nature of Problem	No. of Variables	No. of Equality Constraints	No. of Inequality Constraints
1	Quadratic Objective and Constraints	4	0	3
2	No. 11 of [4]	5	0	3
3	No. 5 of [21]	8	0	23
4	No. 18 of [4]	15	0	5
5	No. 6 of [21]	17	10	8
6	No. 10 of [4]	5	0	10
7	Alkylation Problem from [27]	7	0	14

TABLE 2 . Characteristics of Test Problems

In solving problem 5 by the interior penalty code, the equality constraints were used to eliminate 10 variables, leaving an inequality constrained problem. All problems were solved on the UNIVAC 1108 at Case Western Reserve University. Results are shown in Table 3.

Statistic	Penalty	GRG	Reduction Factor Penalty/GRG
One Dimensional Searches	495	94	5.3
Function Calls	3773	1124	3.4
Gradient Calls	539	99	5.4
Equivalent Function Calls*	8645	2023	4.5

TABLE 3. Comparison of GRG and Interior Penalty Codes

*Equivalent Function Calls ≡ Function Calls + N·Gradient Calls
where N = No. of variables.

GRG required far fewer one-dimensional searches, function evaluations and gradient evaluations. While some of this reduction was offset by the requirement of matrix inversion and solution of nonlinear equations in GRG, GRG produced more accurate solutions for most of the problems. Computation times were a few seconds or less, and differences in computation times (of the order of tenths of a second) could not be estimated accurately due to the masking effects of multiprogramming.

6.2 Solution of Himmelblau Test Problems

The first twenty-four problems specified in Appendix A of reference [4], which includes all of the problems with equality or inequality constraints or element value bounds, have also been solved with GRG*. Several were solved with more than one starting point and one (number 22) with four sets of parameter values. Table 4 shows the results of solving these problems on an IBM 370/145 at Cleveland State University.

In this table, the Newton Average is the total number of iterations of the quasi-Newton method (see section 5.4) divided by the number of times solution of a nonlinear system was attempted, i.e., the number of calls to subroutine NEWTON. The Colville standard time is obtained by dividing the execution times by 77.83, the time in seconds required to run the Colville standard timing program (see [4]) on the 370/145. These numbers provide some means (admittedly imperfect) for other investigators, using different computers, to compare results.

*We thank Professor David Himmelblau for providing us with a card deck for these problems.

The current version of GRG solved all but one of these problems (number 6), in the sense that at least a local minimum was found. In all but two of the problems (6 and 14) the final objective values attained by GRG starting with the initial points specified in [4] either matched the solutions specified in [4], to at least one part in one thousand, or were better than the solutions given in [4].

In problem number 6, using the starting point specified in [4], GRG reached a point with function value -1865.98 compared to -1910.361 given in [4]. However the constraints were satisfied within 1 part in 10^{12} using GRG but only within 1 part in 20 for the solutions given in [4]. Using a different starting point (x = 0) GRG does reach an objective value of -1910.22 with constraints satisfied to 1 part in 10^{10}.

Problem number 14 "contained a myriad of local optima of many different values." Depending on the starting point used, GRG generated several different solutions. Using the initial feasible starting point in [4], GRG reached a minimum of 261,350 compared to 250800 in [4]. From the nonfeasible starting point GRG attains a minimum of 260,508. Starting from $x_i = 0$, $i \neq 4$, $x_4 = 2000$ GRG reaches a value 251,786.

These results were attained using the default values for all parameters in GRG, finite difference approximations for derivatives, quadratic extrapolation for basic variables, and double precision floating point computations. We also examined the effects of changing the EPSTOP parameter (see section 5.1) from 10^{-4} to 10^{-3}. This reduced run times for the Himmelblau problems; only slightly for most problems, up to 25% for some. Two problems, however, stopped significantly short

Problem Number	N, M, NEQ	Best function value reported	Best function value with GRG	Function and gradient evaluations	Equivalent function evaluations	One dimensional searches	Newton average	Execution time (sec)	Colville standard Time
1	2,2,1	1.393	1.393	25,4	33	3	0.54	0.10	0.0013
2	2,0,0	0.0	6.0×10^{-14}	177,25	227	25	0	0.44	0.0057
3	2,3,0	58.903	58.903	169,17	203	16	3.94	1.04	0.013
4	10,3,3	-47.761	-47.720	77,17	247	16	0	3.81	0.049
4A	10,3,3	-47.761	-47.755	158,16	318	15	2.0	3.98	0.051
5	3,2,2	961.715	961.715	39,7	60	6	0.5	0.24	0.0031
6	45,16,16	-1910.361	-1865.98	229,50	2479	49	0	227.26	2.920
7	3,7,0	-1162.04	-1162.03	130,17	151	16	0.82	2.75	0.035
8	4,0,0	0.0	1.0×10^{-7}	255,43	427	43	0	1.32	0.017
9	4,1,0	0.0075	0.0075	89,19	165	18	0	18.26	0.235
10	5,10,0	-32.349	-32.349	63,9	108	9	0	1.53	0.020
11	5,3,0	-30,665.5	-30,665.5	16,6	46	5	1.0	0.21	0.0027
11A	5,3,0	-30,665.5	-30,665.5	44,5	69	4	1.92	0.27	0.0035
12	5,21,0	-1.905	-1.9051	48,6	78	5	1.71	1.01	0.013
13	5,3,0	-5,280,254	-5,280,338	19,6	49	5	0.33	0.16	0.0021
14	6,4,0	255,303.5	261,350.5	118,19	232	18	0.38	2.93	0.038
14A	6,4,0	266,754.0	260,507.8	102,14	186	13	0.53	2.24	0.029
15	6,4,4	8927.59	8927.57	172,17	274	16	1.75	2.41	0.031
16	9,13,0	-0.8660	-0.86604	244,18	406	17	2.28	4.39	0.056
17	10,0,0	-45.778	-45.778	32,5	82	5	0	1 72	0.022
18	15,5,0	32.386	32.349	564,42	1194	41	2.96	18.65	0.240
18A	15,5,0	32.386	32.348	399,31	864	30	2.04	13.51	0.174
19	16,8,8	-244.90	-244.90	162,37	754	36	0	38.55	0.495
20	24,20,14	0.05700	0.05566	200,31	744	29	1.00	10.85	0.139
21	3,0,0	0.0	0.0	6,2	12	1	0	1.90	0.025
22,22A	6,4,0	0.0156	0.0156	8,7	50	6	0	0.28	0.0036
22B,22C	6,4,0	4.070	3.1358	58,9	112	8	1.39	0.56	0.0072
23	100,12,0	-1732.	-1733.3	239,41	4339	40	0.03	570.37	7.328
24	2,2,0	1.0	1.0	26,4	34	3	1.17	0.08	0.0010

TABLE 4. Results of Solving Himmelblau Problems

of optimality. This suggests a strategy in which a loose stopping tolerance is met, the tolerance is decreased, and the procedure repeated until no significant change in objective or x values is detected. This had not yet been implemented.

Table 5 shows the results of using analytic derivatives for problems 6 and 23. The constraints for these problems were linear and hence analytic derivatives were easily obtained. For all practical purposes the number of iterations, etc. was independent of the gradient computation. Run times are dramatically shorter when the gradients are analytically computed. Even greater reductions could have been obtained by coding the partial derivative subroutines PARSH to evaluate the constraint gradients (which are constant) once only.

In order to obtain some measure of the impact of performing the floating point computations in double precision, the Colville program was modified to also run in this mode. As a result of this change, the average execution time increased from 77.83 to 109.89 seconds or 41%. Earlier versions of GRG used only single precision computations and occasionally failed to locate a relative minimum. Again, in the interest of robustness, execution speed has been sacrificed in the current version.

Problem Number	Time with Finite Diff. Derv.	Time with Analytic Deriv.	Percent Increase
6	227.26	66.08	244
23	570.37	103.64	451
Totals	797.73	169.72	368

TABLE 5. Analytic versus Finite Difference Derivatives

6.3. Comparison with Two Other NLP Codes

The test problems of Table 4 have also been solved* by two other codes--Abadie's 1975 GREG (a GRG code) and a code developed by Newell and Himmelblau [22] implementing an "exact" exterior penalty method, or "Method of Multipliers"--as well as by an earlier version of the code described here. All problems were solved on the CDC-6600 at the University of Texas at Austin. Computation times (in seconds) are shown in Table 6, where F indicates a failure to solve the problem and a dash indicates that no attempt to solve was made. The totals row is the sum of times for those 17 problems which were solved by all 3 codes.

The GRG code described here compares well, especially since the current version has solved all but problem 21. In addition, our code used finite difference approximations to derivatives, while the other two computed them analytically. Solution times are consistently small, and the code does well on the larger problems 18 and 20. Its advantage in total time over the other two codes is due to a distinct superiority on a few problems. In fact, the Newell code was fastest in 10 problems, ours in 5 and GREG in 2.

---

*We are grateful to Professor David Himmelblau for running these tests.

Problem number	Abadie GREG 1975	Lasdon-Waren GRG Code	Newell method of multipliers
1	.11	.09	.01
2	.21	.46	.02
3	.18	.15	.07
4	1.07	.48	.88
5	.18	.16	.01
7	1.70	1.58	.45
8	.84	.77	.16
9	27.59	1.55	5.11
10	.31	.41	.16
11f	---	.12	.64
11nf	.21	---	.62
12	2.58	F	11.7
13	.17	.11	---
14	.56	.46	.58
15	.52	.64	.22
16	F	F	.19
17	.21	.29	.08
18	3.51	2.36	13.15
19nf	.97	F	3.96
20	16.52	1.00	6.06
21	30.28	F	6.97
22A	.16	.18	.85
22C	.29	.25	1.04
24	.15	.09	.04
TOTALS	54.22	10.92	28.89

TABLE 6

Computation Times (CD6600 Sec.) for Himmelblau Problems

6.4. Comparsion of Linear vs. Quadratic Extrapolation of Basic Variables

As discussed in Section 4, either linear or quadratic extrapolation can be used in subroutine REDOBJ to estimate initial values for the basic variables. The quadratic estimates take more time and storage to compute but should reduce the number of Newton iterations required. To assess the relative merits of these two options all test problems of section 6.3 requiring any Newton iterations were solved both ways. Only for problem number 18 did quadratic extrapolation take more equivalent function evaluations although the number of Newton iterations remained the same. In all other problems quadratic extrapolation performed as well as or better than linear extrapolation. On an overall basis, the number of equivalent function evaluations decreased by 5%, Newton iterations decreased 16% and execution time decreased by 5%. In another series of tests, the 7 problems of section 6.1, plus 2 others of 15 and 7 variables were solved. All are highly nonlinear. Using quadratic versus linear extrapolation yielded the following cumulative results: one-dimensional searches increased 19%, Newton iterations decreased 43%, and equivalent function evaluations decreased 20%. We conclude that quadratic extrapolation can reduce function evaluations significantly (especially for highly nonlinear problems), and should reduce computation time appreciably when GCOMP calls account for a large fraction of the overall computational effort.

7. Conclusions and Future Work

The results presented here indicate that GRG, as implemented in this code, is an efficient and reliable way to solve small to moderate size NLP problems. Abadie's GRG codes have also received extensive testing (e.g. in [4] and [23]). Our results support the conclusions reached by these authors, namely that GRG is a leading contender among NLP algorithms. Since small, dense, highly nonlinear problems have almost infinite variety, it is unlikely that any one algorithm will be best for all problems. Modern penalty "augmented Lagrangian" algorithms (see [24]) are among the chief competitors, and comparative tests are planned in the near future.

The best test of a code is to attempt to solve a wide variety of real problems. To facilitate such testing, we are offering the code to persons in nonprofit research institutions for a preparation charge of $55. Both user and system documentation will be provided. Interested parties are urged to contact the authors.

Future work will attempt to extend GRG to large, sparse, "mostly linear" NLP problems of general (nonseparable) nature. A number of large problems of special form, e.g., electric power distribution [25] and hydroelectric scheduling [26], have already been successfully solved by variants of GRG. To our knowledge, no general purpose GRG code capable of solving large problems exists today. We believe that few new ideas are required to construct such a code; a union of existing LP technology and some of the ideas presented here should suffice. Work on this project has recently begun.

REFERENCES

1. F. Glover, D. Karney, D. Klingman, "A Comparison of Computation Times for Various Starting Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science 20, No. 5 (1974), 793-814.

2. A. Geoffrion and G. Graves, "Multicommodity Distribution System Design by Benders Decomposition," Management Science, 20, No. 5 (1974), 822-844.

3. Harwell Subroutine Library Descriptions (various authors), Computer Science and Systems Division, Atomic Energy Research Establishment, Harwell, Oxfordshire, England.

4. D. M. Himmelblau, Applied Nonlinear Programming, McGraw-Hill Book Co., 1972.

5. J. W. Bandler, "Availability of Internal Reports," Group on Simulation, Optimization and Control, McMaster University, Hamilton, Ontario, Canada, L8S 4L7.

6. G. P. Mc Cormick, "A Mini-Manual for Use of the SUMT Computer Program and the Factorable Programming Language," Technical Report SOL-74-15, Systems Optimization Laboratory, Dept. of Operations Research, Stanford, Ca., 1974.

7. J. B. Rosen and Steve Wagner, "The GPM Nonlinear Programming Subroutine Package. Description and User Instructions," Technical Report 75-9, Department of Computer and Information Sciences, University of Minnesota, May 1975.

8. J. Abadie and J. Carpentier, "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints," in Optimization R. Fletcher, Ed., Academic Press, 1969, 37-47.

9. J. Abadie, "Application of the GRG Algorithm to Optimal Control Problems," in Nonlinear and Integer Programming, J. Abadie, Ed., North Holland Publishing Co., 1972, 191-211.

10. L. S. Lasdon, R. Fox and M. Ratner, "Nonlinear Optimization Using the Generalized Reduced Gradient Method," Tech. Memo. No. 325, Department of Operations Research, Case Western Reserve University, October, 1973.

11. D. R. Heltne and J. M. Liitschwager, "Users Guide for GRG 73" and "Technical Appendices to GRG 73," College of Engineering, University of Iowa, Sept. 1973.

12. C. Cohen, "Generalized Reduced Gradient Technique for Non-linear Programming--User Writeup," Vogelback Computing Center, Northeastern University, Feb. 1974.

13. D. Gabay and D. Luenberger, "Efficiently Converging Minimization Methods Based on the Reduced Gradient," Internal report, Department of Engineering-Economic Systems, Stanford University.

14. R. Fletcher, "A New Approach to Variable Metric Algorithms," Computer Journal, 13 (1970), 317-322.

15. D.F. Shanno and K. H. Phua, "Inexact Step Lengths and Quasi Newton Methods," working paper, University of Toronto, 1974.

16. D. F. Shanno, A. Berg and G. Cheston, "Restarts and Rotations of Quasi-Newton Methods," in Information Processing 74, North Holland Publishing Co., 1974, 557-561.

17. D. Goldfarb, "Extension of Davidons Variable Metric Method to Maximization Under Linear Inequality and Equality Constraints," SIAM J. Appl. Math. 17, No. 4, July 1969.

18. A. Buckley, "An Alternate Implementation of Goldfarbs Minimization Algorithm," Report No. T.P. 544, AERE, Harwell, England.

19. R. Fletcher and C. M. Reeves, "Function Minimization by Conjugate Gradients," British Computer J. 7 (1964), 149-154.

20. L. S. Lasdon, A. D. Waren, A. Jain and M. Ratner, "Design and Testing of a GRG Code for Nonlinear Optimization," Tech. Memo 20.353, Operations Research Department, Case Western Reserve University, Cleveland, Ohio, March 1975.

21. L. S. Lasdon, R. Fox, M. W. Ratner, "An Efficient One-Dimensional Search Procedure for Barrier Functions," Mathematical Programming 4 (1973), 275-296.

22. J. S. Newell and D. M. Himmelblau, "A New Method for Nonlinearly Constrained Optimization," AICHE J. 21, No. 3 (May 1975), 479-486.

23. A. R. Colville, "A Comparative Study of Nonlinear Programming Codes," IBM New York Scientific Center Report 320-2949, 1968.

24. R. Fletcher, "An Ideal Penalty Function for Constrained Optimization," Internal Report C.S.S. 2, Computer Sciences and Systems Division, Atomic Energy Research Establishment, Harwell, England.

25. A. Sassoon and H. Merrill, "Some Applications of Optimization Techniques to Power Systems Problems," Proceedings of the IEEE 62, No. 7 (July 1974), 959-975.

26. C. R. Gagnon et al., "A Nonlinear Programming Approach to a Very Large Hydroelectric System Optimization," Mathematical Programming, 6 (1974), 28-41.

27. Bracken, J. and Mc Cormick, G.P., "Selected Applications in Nonlinear Programming," John Wiley and Sons, N.Y., 1968.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: SOL-76-3 (14)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle): DESIGN AND TESTING OF A GENERALIZED REDUCED GRADIENT CODE FOR NONLINEAR PROGRAMMING. (6)	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
7. AUTHOR(s): L.S. LASDON, Arvind JAIN, A.D. WAREN, Margery RATNER (10)	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0267, N00014-75-C-0865, N00014-75-C-0240 (Case) (15)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Department of Operations Research, Stanford University, Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-047-064, NR-047-143, NR-047-105 (16)
11. CONTROLLING OFFICE NAME AND ADDRESS: Operations Research Program Code 434, Office of Naval Research, Arlington, Virginia 22217 (11)	12. REPORT DATE: Feb 1976 (12) 51p. / 13. NUMBER OF PAGES: 47
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

GRG
NONLINEAR PROGRAMMING
OPTIMIZATION SOFTWARE
LARGE SCALE OPTIMIZATION

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of this paper is to describe a Generalized Reduced Gradient (GRG) algorithm for nonlinear programming, its implementation as a FORTRAN program for solving small to medium size problems, and some computational results. Our focus is more on the software implementation of the algorithm than on its mathematical properties. This is in line with the premise that robust, efficient, easy to use NLP software must be written and made accessible if nonlinear programming is to progress, both in theory and in practice.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

K08765